



Università degli Studi di Parma

Dipartimento di Ingegneria e Architettura

Sistemi operativi e in tempo reale - a.a. 2022/23

Protocolli di accesso a risorse condivise per task in tempo reale

prof. Stefano Caselli



Dove sono finite le risorse?

- ❑ La contesa per le risorse influenza il comportamento e la schedulabilità dei job
 - L'accesso non regolato alle risorse produce errori inaccettabili e deve essere compatibile con i vincoli di tempo reale
- ❑ Sono stati sviluppati protocolli di accesso alle risorse che attenuano gli effetti della contesa e definiti metodi per tenerne conto dal punto di vista delle garanzie real-time
- ❑ Nei sistemi clock-driven l'accesso alle risorse è garantito all'interno di slice, e quindi è pianificato a priori
- ❑ → problema rilevante per i sistemi priority-driven !



Modello delle risorse

- ❑ Unità di risorse riusabili in modo sequenziale
- ❑ Richiedono accesso esclusivo per l'utilizzo; una volta assegnate ad un job possono essere utilizzate da altri job solo dopo che sono state rilasciate
- ❑ Presuppongono controllo dell'accesso mediante primitive del SO: ad es. *lock(x)* e *unlock(x)* oppure *wait(sem)* e *signal(sem)*



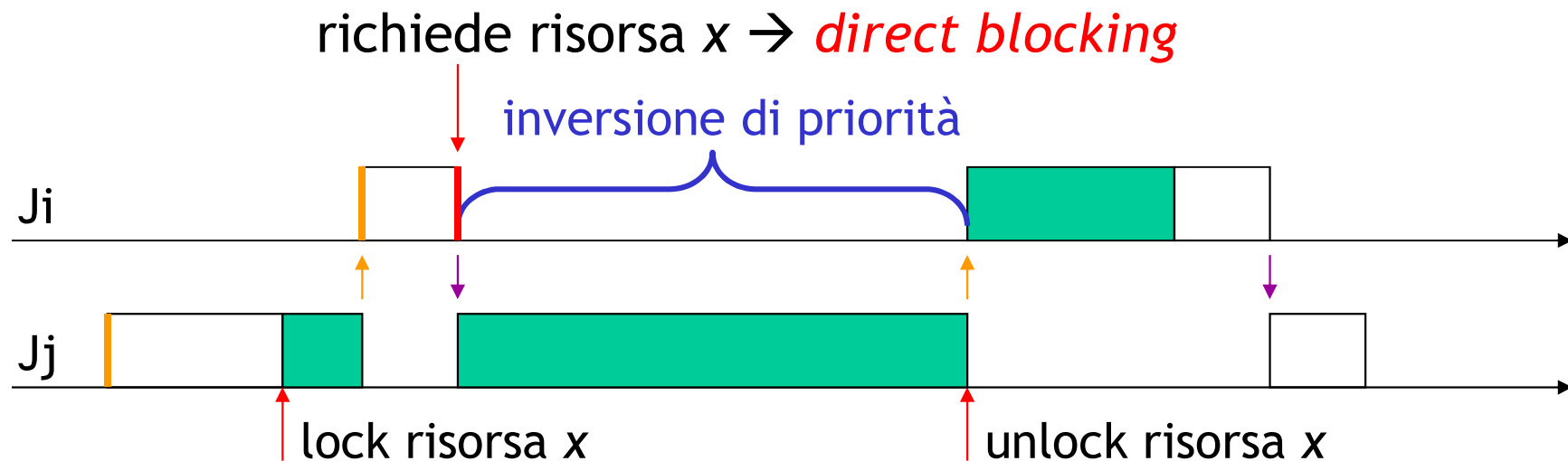
Inversione di priorità

- In presenza di sincronizzazioni si può verificare una situazione di *inversione della priorità*, in cui un job a priorità inferiore ne *blocca* uno a priorità più elevata (ad esempio su un semaforo mutex)



Inversione di priorità

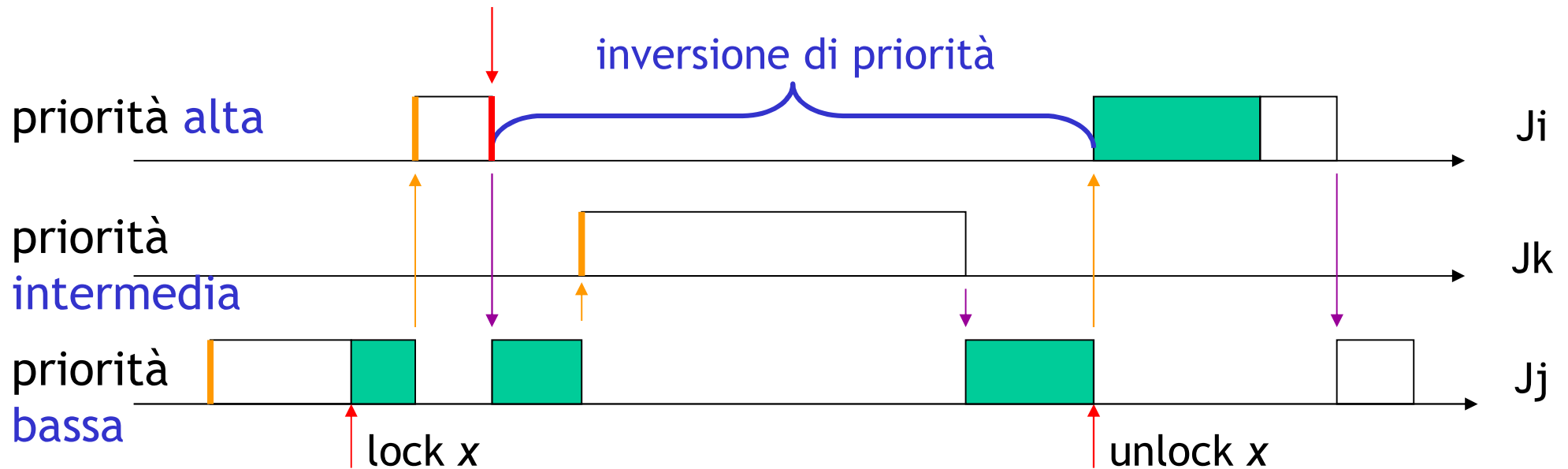
- Un job ad alta priorità è ritardato nell'esecuzione da parte di un job a priorità più bassa
 - → la relazione di priorità è invertita!
- J_i job ad alta priorità; J_j job a bassa priorità:





Inversione di priorità per un tempo illimitato

- La durata della inversione di priorità non è funzione solamente del tempo necessario al job a bassa priorità per eseguire la sezione critica!
- *Unbounded priority inversion*





Tempo di risposta di un job

- ❑ Nel caso peggiore è determinato da:
 - Tempo di esecuzione del job
 - Preemption subita da altri job a priorità maggiore
 - Tempo di blocco: ritardo subito nello stato bloccato
- ❑ Nel caso più favorevole il tempo di blocco è una funzione semplice dei ritardi subiti mentre altri job sono in sezione critica
- ❑ Se così non è, il tempo di blocco è difficile da calcolare



Inversione di priorità

- In presenza di sincronizzazioni si può verificare una situazione di *inversione della priorità*, in cui un job a priorità inferiore ne *blocca* uno a priorità più elevata (ad esempio su un semaforo mutex)
- Il job a priorità inferiore può a sua volta essere interrotto da uno a priorità intermedia
- \Rightarrow Occorre *innalzare temporaneamente la priorità di un job* a quella più alta tra tutti i job che esso blocca (*priority boosting*)

Tecniche per evitare inversione di priorità incontrollata



- ❑ Sezioni critiche non revocabili
- ❑ Protocollo Priority Inheritance
- ❑ Protocollo Priority Ceiling



Sezioni critiche non revocabili

- ❑ L'approccio è denominato protocollo *NPCS* (*Non-Preemptive Critical Sections*) (Mok, 1983)
- ❑ Caratteristiche:
 1. Quando un job *richiede* una risorsa, essa è libera per definizione e la risorsa gli viene allocata
 2. Quando un job *detiene* una risorsa, esso esegue ad una priorità superiore a tutti gli altri job e *non può subire preemption*
- ❑ Nota: con NPCS non si può verificare deadlock



Sezioni critiche non revocabili

- Teorema: Con il protocollo NPCS un job J_i può essere bloccato una sola volta
- In un sistema RT *a priorità fissa* con n task, il tempo di blocco massimo B_i che un task periodico τ_i può subire a causa della contesa sulle risorse è pari alla *più lunga sezione critica* di *tutti* i task a priorità inferiore
- Sia ξ_k la più lunga sezione critica del task τ_k
- Ordinando i task con priorità decrescente, si ha:

$$B_i = \max_{i+1 \leq k \leq n} \xi_k$$



Sezioni critiche non revocabili

- Con EDF, il job di un task τ_i con deadline relativa D_i può subire blocco solo da job J_k di task con deadline relativa $D_k > D_i$
- Sia ξ_k la più lunga sezione critica del task τ_k
- Il massimo tempo di blocco B_i che un task periodico τ_i può subire a causa della contesa sulle risorse è ancora pari alla più lunga sezione critica di tutti i task con deadline relativa maggiore:

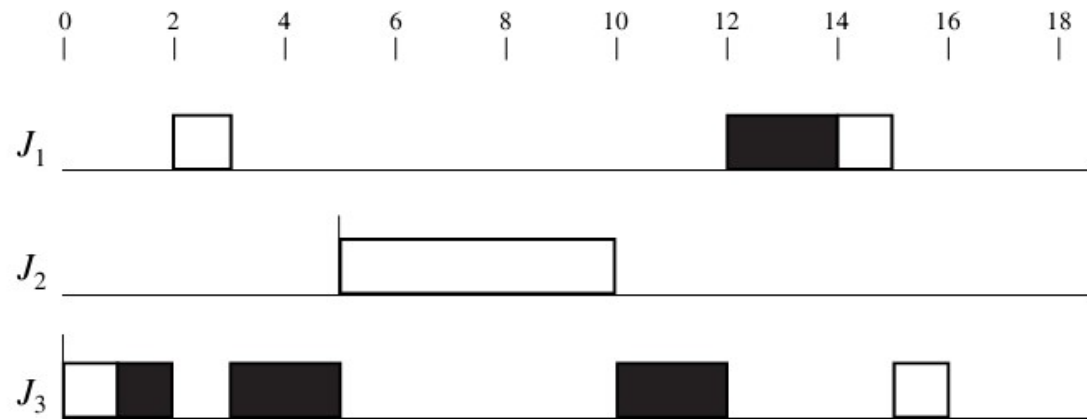
$$B_i = \max_{i+1 \leq k \leq n} \xi_k$$

- ove però i task sono ordinati in base alle deadline relative crescenti ($i < j$ se $D_i < D_j$)

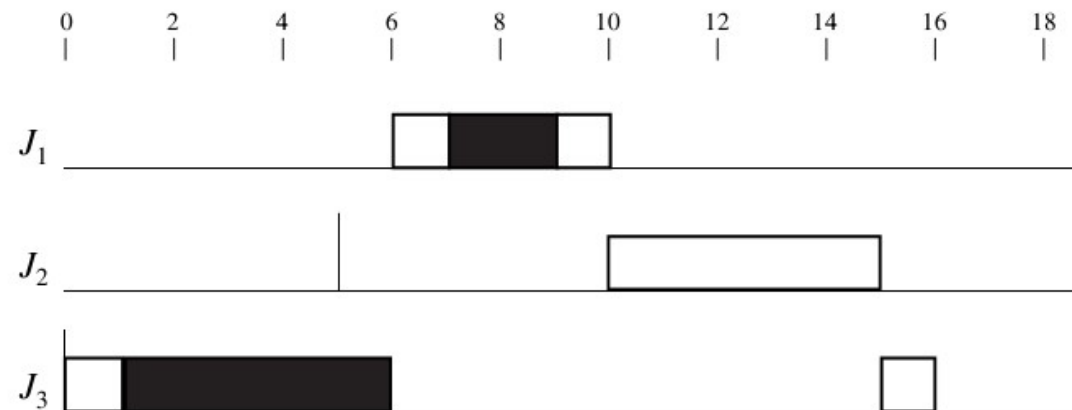
Esempio con sezioni critiche non revocabili



- Situazione di inversione di priorità:



- Con NPCS:





Sezioni critiche non revocabili

- ❑ Caratteristiche del protocollo NPCS:
- ❑ Semplice, anche con numero arbitrario di risorse
- ❑ Trasparente al programmatore! (ci pensa il SO/supporto runtime)
- ❑ Funziona bene *se le sezioni critiche sono tutte brevi*, anche nel caso di sezioni critiche numerose e conflitto tra molti task
- ❑ Svantaggio: un job può essere bloccato *da un qualunque job a priorità inferiore* anche se non c'è alcuna contesa/ conflitto tra loro
- ❑ Grave se le sezioni critiche non sono tutte brevi!



Protocollo priority inheritance

- ❑ [Sha et al., 1990]
- ❑ Protocollo applicabile con *qualunque algoritmo priority-driven*, con priorità statiche o dinamiche, *preemptive* (→ EDF, RM, DM)
- ❑ Trasparente al programmatore
- ❑ Previene inversioni di priorità di durata incontrollata
- ❑ Realizzazione relativamente semplice
- ❑ Non risolve tutti i problemi:
 - Possibili catene di attesa lunghe, nel caso peggiore
 - Non previene deadlock, quindi attese indefinite



Protocollo priority inheritance: ipotesi

- Priorità nominale assegnata ai job in base al criterio di priorità adottato dall'algoritmo
- FCFS tra job di eguale priorità
- J_1, J_2, \dots, J_n con priorità nominali decrescenti
- Sezioni critiche anche annidate, con annidamento regolare
- Accesso a sezioni critiche con semafori binari o primitive lock/unlock:

lock(Ris) \leftarrow ----- sez critica ----- \rightarrow unlock(Ris)



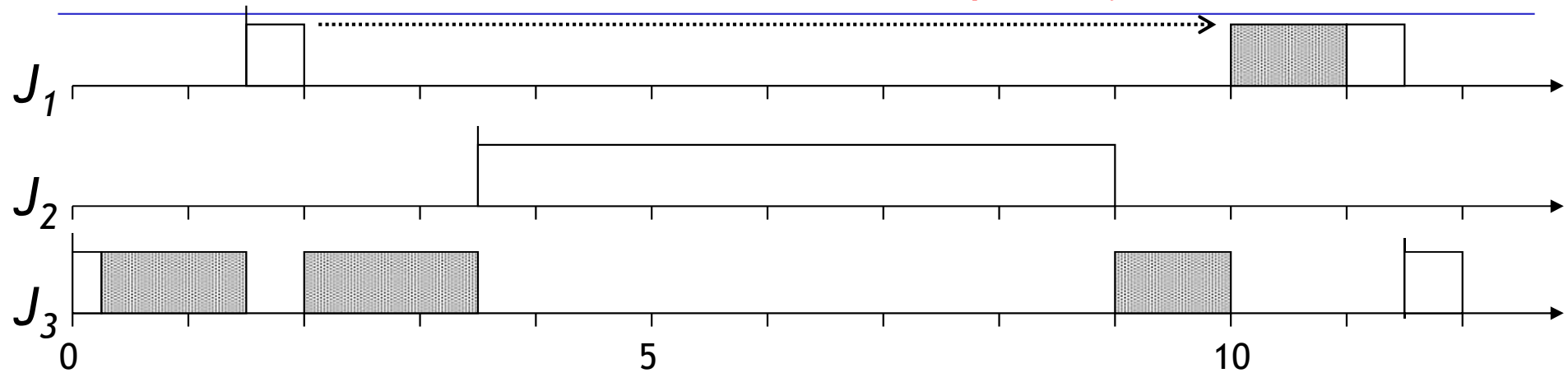
Protocollo priority inheritance

- Se un job J_a si blocca all'ingresso di una sezione critica esso *trasmette la propria priorità* a J_b che la detiene
- J_b *esegue alla priorità massima* tra quelle dei job che sta bloccando a causa delle risorse che detiene
- All'uscita dalla sezione critica il job J_b è riportato alla sua priorità naturale
- L'ereditarietà della priorità è transitiva
- Due situazioni di blocco per un job ad alta priorità:
 - Perché la sezione critica a cui vuole accedere è occupata da un job a priorità inferiore → *blocco diretto*
 - Perché un job a bassa priorità ha ereditato una priorità maggiore in sezione critica → *blocco push-through*

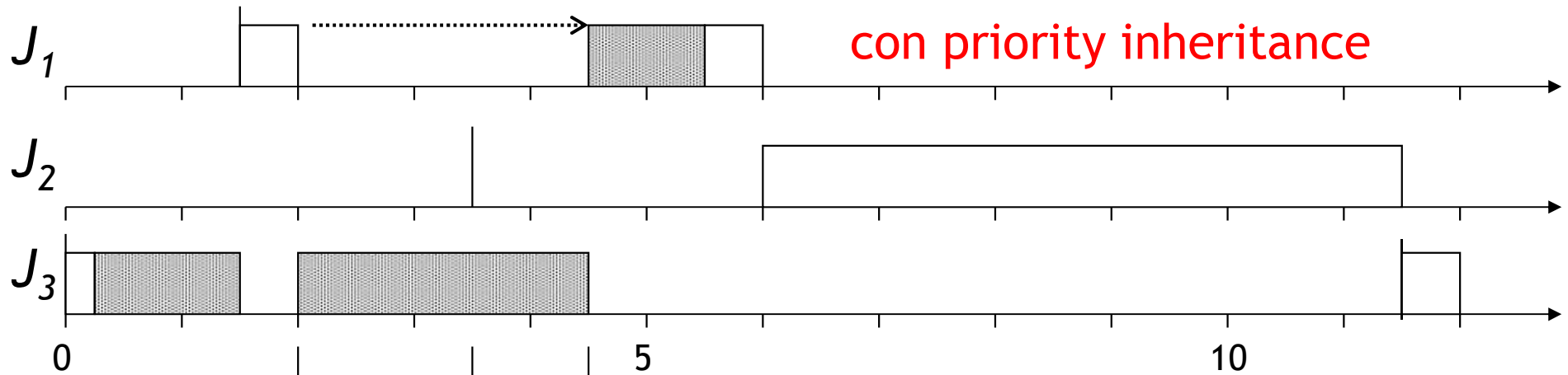
Priority Inheritance



senza priority inheritance



$$\pi_1 > \pi_2 > \pi_3$$



con priority inheritance

J_3 blocca J_2 (blocco push-through)

J_3 blocca J_1 (blocco diretto)

Accesso a risorse condivise

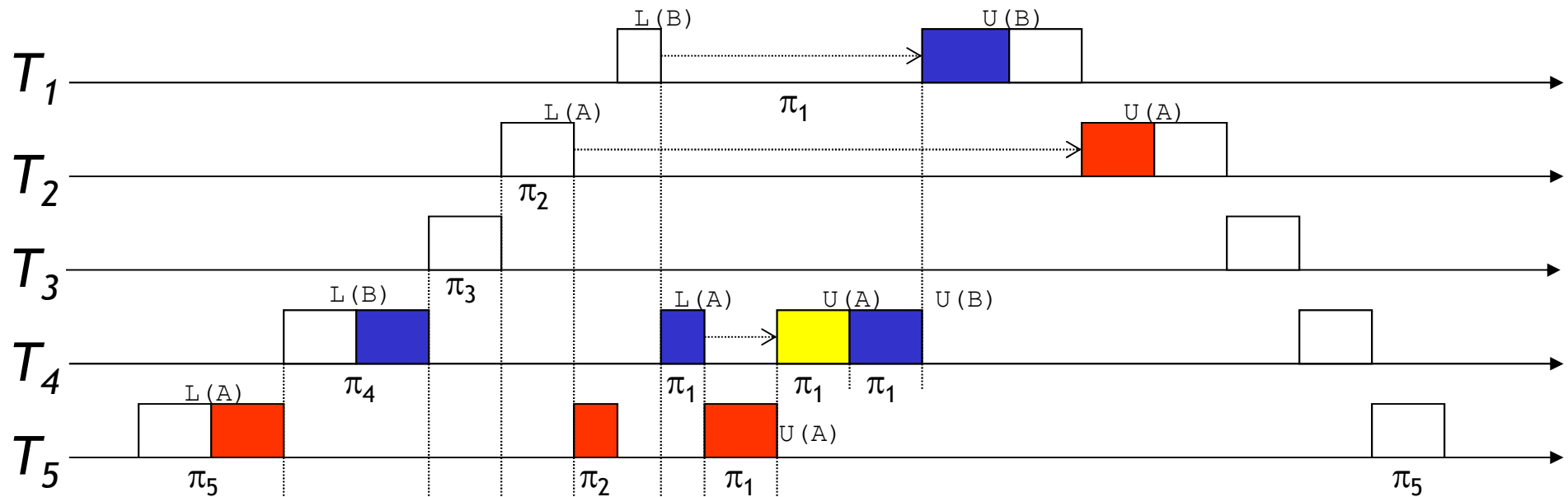
$$Pri(J_3) = \pi_1$$



Priority Inheritance

- Teoremi:
 - T1: Un job J_k può essere bloccato da un job a priorità inferiore J_l *una sola volta*
→ Nel caso peggiore J_k può essere bloccato da ciascun job a priorità inferiore su una *sezione critica diversa*
 - T2: Se ci sono m lock o semafori su cui J_k può dover attendere *a causa di blocco diretto o indiretto*, J_k può essere bloccato al più m volte
 - T3: Un job J_k può essere bloccato al più per la durata di $\min(n, m)$ sezioni critiche, dove n è il numero di job a priorità inferiore di J_k ed m è il numero di lock o semafori su cui J_k può attendere *per blocco diretto o indiretto*
-

Protocollo Priority Inheritance (PIP) - Esempio



$$\pi_1 > \pi_2 > \pi_3 > \pi_4 > \pi_5$$

Uso di A

Uso di A e B

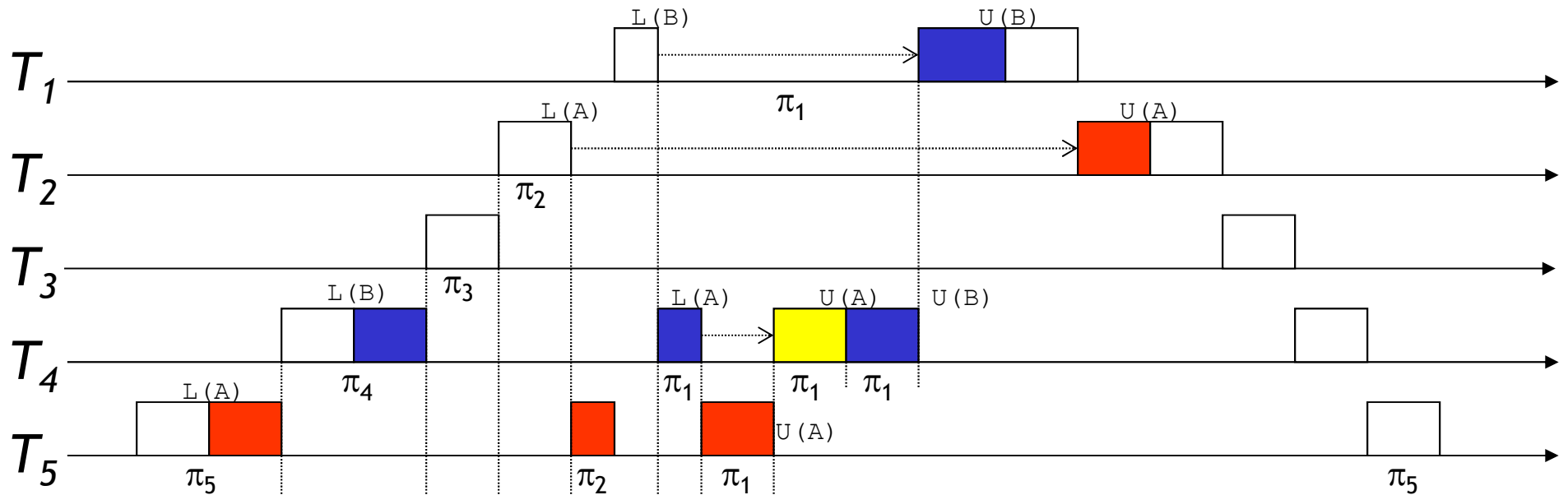
Uso di B



Protocollo PIP: Garanzie

- E' possibile integrare l'effetto della contesa nei bound che caratterizzano l'algoritmo priority-driven considerando il *tempo di blocco massimo* B_i che ciascun task può subire
- RM: $\forall i, 1 \leq i \leq n \quad \sum_{k=1, i} C_k / T_k + B_i / T_i \leq U_{lub}(i)$
- EDF: $\forall i, 1 \leq i \leq n \quad \sum_{k=1, n} C_k / T_k + B_i / T_i \leq 1$
- Occorre calcolare i B_i sulla base dei teoremi precedenti e procedere valutando le garanzie per ciascun task, con costo $O(n)$

Protocollo Priority Inheritance (PIP) - Problema



↑ Cosa accade se qui T_5 esegue $L(B)$?

$$\pi_1 > \pi_2 > \pi_3 > \pi_4 > \pi_5$$



Uso di A



Uso di A e B



Uso di B



Inversione di priorità e garanzia

- ❑ Dati un algoritmo di scheduling priority driven di task periodici ed un protocollo per il controllo della inversione di priorità, come si applicano i test di garanzia?
- ❑ Con algoritmi statici (RM, DM, altro): un job può essere bloccato solamente da job appartenenti a task a priorità inferiore
- ❑ Con EDF vale il teorema di Baker: un job può essere bloccato solamente da job appartenenti a task con deadline relativa maggiore
- ❑ Con algoritmi dinamici diversi da EDF: un job può essere bloccato da job appartenenti a tutti gli altri task



Esercizi su task interagenti

- Ci limitiamo qui a casi privi di annidamenti di sezioni critiche
 - Semplificazione!
 - Non si può verificare deadlock: manca possesso e attesa su risorse condivise
- Es. 1: Determinare la schedulabilità RM dei seguenti task (Bi è il tempo di blocco):

	Ci	Ti	Bi
J1	1	2	1
J2	1	4	1
J3	2	8	0



Esercizi su task interagenti

- Es. 2: Blocking time computation con PIP e NPCCS
- Dati i task periodici $\{J_i\}$, che competono per le sezioni critiche $\{C_j\}$ secondo la seguente tabella:

	C1	C2	C3
J1	1	2	0
J2	0	9	3
J3	8	7	0
J4	6	5	4

- Determinare il *massimo tempo di blocco* e il *numero di blocchi* subiti da ciascun task ($\pi_1 > \pi_2 > \pi_3 > \pi_4$)



Esercizi su task interagenti

- Es. 2: Blocking time computation con PIP e NPCS
- Dati i task periodici $\{J_i\}$, che competono per le sezioni critiche $\{C_j\}$ secondo la seguente tabella:

	C1	C2	C3	Ni Bi	Ni Bi	Ni Bi
J1	1	2	0	2 17	1 9	1 9
J2	0	9	3	2 13	1 8	1 8
J3	8	7	0	1 6	1 6	1 9
J4	6	5	4	0 0	0 0	1 9
				PIP	NPCS	NPCS non EDF

- Massimo tempo di blocco B_i e numero di blocchi subiti N_i da ciascun task ($\pi_1 > \pi_2 > \pi_3 > \pi_4$ per RM e EDF/Baker)



Esercizi su task interagenti

- Es. 2bis: Blocking time computation con PIP e NPCS
- Dati i task periodici $\{J_i\}$, che competono per le sezioni critiche $\{C_j\}$ secondo la seguente tabella:

	C1	C2	C3	Ni Bi		Ni Bi		Ni Bi	
J1	1	2	0	2	17	1	41	1	41
J2	0	9	3	2	49	1	41	1	41
J3	8	7	0	1	41	1	41	1	41
J4	6	5	<u>41</u>	0	0	0	0	1	9
				PIP		NPCS		NPCS non EDF	

- $\pi_1 > \pi_2 > \pi_3 > \pi_4$ per RM e EDF/Baker



Esercizi PIP e NPCS

- Es. 3: Blocking time computation
- Dati i task periodici $\{J_i\}$, che competono per le sezioni critiche $\{C_j\}$ secondo la seguente tabella:

	C1	C2	C3	C4	
J1	1		3;2		// J1 accede 2 volte a C3
J2		1	1;2	3	// per J2 caso peggiore =2
J3	2			1	
J4	1	1		4	

- Determinare il massimo tempo di blocco e il numero di blocchi subiti da ciascun task sia con NPCS che con PIP
-



Esercizi PIP e NPCS

- Es. 4: Blocking time computation
- Dati i task periodici $\{J_i\}$, che competono per le sezioni critiche $\{C_j\}$ secondo la seguente tabella:

	C1	C2	C3	C4
J1	1		3;2	
J2		1	1;2	
J3	1			80
J4	1	2		100

- Determinare il massimo tempo di blocco e il numero di blocchi subiti da ciascun task sia con NPCS che con PIP



Esercizi PIP e NPCS

- Es. 4: Blocking time computation
- Dati i task periodici $\{J_i\}$, che competono per le sezioni critiche $\{C_j\}$ secondo la seguente tabella:

	C1	C2	C3	C4
J1	1		3;2	
J2		1	1;2	
J3	1			80
J4	1	2		100

Bi	Ni
3	2
3	2
100	1
0	0

Bi	Ni
100	1
100	1
100	1
0	0

*Perché J1 e J2
sono ritardati da
J3 e J4, lenti?
⇒ NPCS poco
gradito ...*

- Determinare il massimo tempo di blocco e il numero di blocchi subiti da ciascun task sia con NPCS che con PIP




Esercizi PIP e NPCS

- Es. 5: Blocking time computation
- Dati i task periodici $\{J_i\}$, che competono per le sezioni critiche $\{C_j\}$ secondo la seguente tabella:

	C1	C2	C3
J1	1	0	0
J2	0	20	30
J3	0	0	0
J4	4	2	2
J5	2	3	3

- Determinare il massimo tempo di blocco e il numero di blocchi subiti da ciascun task sia con NPCS che con PIP

Protocollo priority ceiling

- ❑ Viene attribuito un valore di *ceiling di priorità* pari alla massima priorità dei task che possono accedere alla risorsa
 - ❑ Un job che accede ad una risorsa acquisisce una priorità pari al ceiling
 - ❑ Nella analisi di schedulabilità occorre ancora considerare il tempo di blocco massimo che ciascun task può subire
 - ❑ Il protocollo Priority Ceiling *previene il deadlock*; inoltre ciascun job può essere *bloccato una sola volta*
 - ❑ PCP si presta ad essere integrato con algoritmi a *priorità statica*, mentre l'integrazione con algoritmi a priorità dinamica è complicata (il ceiling delle risorse cambia!)
- 



Task interagenti nei sistemi real-time

- Test $O(1)$ per NPCS, PIP e PCP:

$$\sum_{i=1,n} \underline{C_i/T_i} + \max_{i=1,n-1} (B_i/T_i) \leq U_{lub}(n)$$

- (i tempi di blocco B_i cambiano tra NPCS, PIP, PCP)
- Tutto assieme, più semplice ma più restrittivo
- Alcuni «detti» comuni della programmazione RT:
 - The best lock is the one that you don't need!
 - Don't share if you can avoid it
 - Make sharing tasks at the same priority if you can, or fuse them in single task

Posizione un po' estrema, anche se autorevole

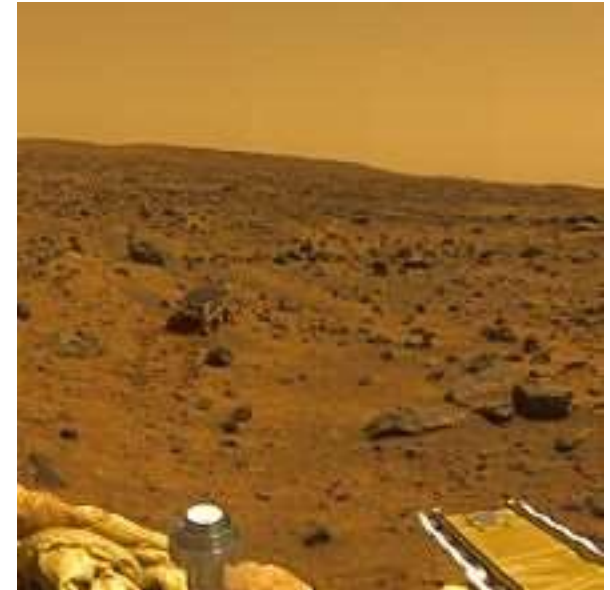


- Philip Koopman (CMU), nel libro «*Better embedded system software*», pag 147:
 - Don't use EDF scheduling:
 - (with EDF or MLF) system behavior is very bad if system load goes above 100%, whereas RMS is better behaved in overload conditions
 - Moreover, EDF and MLF require run-time changes to priorities, which is more complex than the fixed priority approach used by RMS
 - --> *Stay away from EDF and Least Laxity, use RMS if you can*
- Voi siete gli esperti che possono utilizzare in modo competente anche EDF!

But what really happened on Mars, in July 1997?



Sojourner incontra la roccia Yogi



Il rover visto da Pathfinder